View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 242 (2009) 35–48

www.elsevier.com/locate/entcs

A Basic Parallel Process as a Parallel Pushdown Automaton

J.C.M. Baeten[1] P.J.L. Cuijpers[1] P.J.A. van Tilburg[1]

Division of Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We investigate the set of basic parallel processes, recursively defined by action prefix, interleaving, **0** and **1**. Different from literature, we use the constants **0** and **1** standing for unsuccessful and successful termination in order to stay closer to the analogies in automata theory.
We prove that any basic parallel process is rooted branching bisimulation equivalent to a regular process communicating with a bag (also called a parallel pushdown automaton) and therefore we can regard the bag as the prototypical basic parallel process.
This result is closely related to the fact that any context-free process is either rooted branching bisimulation equivalent or contrasimulation equivalent to a regular process communicating with a stack, a result that is the analogy in process theory of the language theory result that any context-free language is the language of a pushdown automaton.

Keywords: automata theory, process algebra, basic parallel process, parallel pushdown automaton

1 Introduction

In this paper, we study the class of basic parallel processes. This class was introduced in [7] as the class of all processes that have a finite guarded recursive specification over the small process algebraic language with **0**, action prefix, choice and parallel composition without communication (just interleaving). More work about this class can be found in e.g. [8,11]. Some results correspond to analogous results in formal language theory, such as the fact that every basic parallel language can be presented as a parallel pushdown automaton (a pushdown automaton not with a stack but with a *bag*, a multiset of variables).

However, there is an important difference between automata theory on the one hand and process algebra (CCS style) on the other hand that has been mostly neglected so far. In an automaton, for instance a non-deterministic finite automaton, any subset of the set of states can be marked as final, and for the definition of the

[1] Email: {j.c.m.baeten,p.j.l.cuijpers,p.j.a.v.tilburg}@tue.nl.

1571-0661 © 2009 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2009.06.012

language of an automaton only sequences that lead from the initial state to a final state count. In this sense, successful termination in an automaton is observable. In process algebra CCS style, the only observables are executions of actions, with $\mathbf{0}$ being the process characterized by allowing no actions at all. Sequential composition can be defined, nevertheless, by having special 'tick' actions that by synchronization turn into internal actions. In process algebra ACP style, observables are action executions and action executions leading to termination. Sequential composition then becomes a basic operator. In both the CCS and ACP approaches, however, termination occurring in a choice context (a terminating state with an outgoing edge) cannot be presented accurately. This can be achieved with the introduction of the $\mathbf{1}$ process (characterizing a process that can only terminate), resulting in a full analogy with automata theory. Using this analogy, we can say that a regular process is the bisimulation equivalence class of a non-deterministic finite automaton, and the set of regular processes is exactly the set of processes given by a finite guarded recursive specification over $\mathbf{0}, \mathbf{1}$, action prefix and choice.

We investigated the set of context-free processes (defined with $\mathbf{1}$) in [4]. There, the addition of $\mathbf{1}$ makes an essential difference: a process can be defined that has unbounded branching, something that cannot be done without $\mathbf{1}$. Furthermore, we established in [4] under what conditions a context-free process can be presented as a pushdown automaton. In this paper, we investigate a similar result for the class of basic parallel processes. For basic parallel processes, the added expressivity is less spectacular (a corollary of our main theorem is that basic parallel processes have bounded branching, even those including $\mathbf{1}$), but still, without $\mathbf{1}$ a bag process expressed as a basic parallel process cannot be tested for being empty. In general, adding $\mathbf{1}$ makes that the theory becomes more challenging, and in our opinion also more interesting.

Another difference between automata theory and process theory is that process theory allows us to make communication explicit and abstract from it modulo branching bisimulation. In a setting with explicit communication, a pushdown automaton can be seen as a regular process communicating with a stack. Since every context-free process can be realized in this way, and the stack is a context-free process itself, we can look upon the stack as the prototypical context-free process. Similarly, we show in this paper that every basic parallel process can be presented as a regular process communicating with a bag, a multiset of data elements. Since the bag is a basic parallel process itself, it can be seen as the prototypical basic parallel process.

Thus, the result of [11] that every basic parallel process can be given by means of a parallel pushdown automaton is given here in an extended setting, with the process $\mathbf{1}$ and with explicit communication.

2 Regular Processes

Before we introduce the basic parallel processes, we first consider the notion of a regular process and its relation to regular languages in automata theory. We start

with the definition of the notion of transition system from process theory. A finite transition system can be thought of as a non-deterministic finite automaton. In order to have a complete analogy, the transition systems we study have a subset of states marked as final states.

Definition 2.1 [Transition system] A *transition system* M is a quintuple $(\mathcal{S}, \mathcal{A}, \rightarrow, \uparrow, \downarrow)$ where:

(i) $\mathcal{S}$ is a set of states,

(ii) $\mathcal{A}$ is an alphabet,

(iii) $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the set of *transitions* or *steps*,

(iv) $\uparrow \in \mathcal{S}$ is the initial state,

(v) $\downarrow \subseteq \mathcal{S}$ is a set of final states.

For $(s, a, t) \in \rightarrow$ we write $s \xrightarrow{a} t$. For $s \in \downarrow$ we write $s\downarrow$. A *finite transition system* or *non-deterministic finite automaton* is a transition system of which the sets $\mathcal{S}$ and $\mathcal{A}$ are finite.

In accordance with automata theory, where a *regular language* is a language equivalence class of a non-deterministic finite automaton, we define a *regular process* to be a bisimulation equivalence class of a finite transition system. Contrary to automata theory, it is well-known that not every regular process has a *deterministic* finite transition system (i.e. a transition system for which the relation $\rightarrow$ is functional). The set of deterministic regular processes is a proper subset of the set of regular processes.

Next, consider the automata theoretic characterization of a regular language by means of a right-linear grammar. In process theory, a grammar is called a *recursive specification*: it is a set of recursive equations over a set of variables. A right-linear grammar then coincides with a recursive specification over a finite set of variables in the Minimal Algebra MA. (We use standard process algebra notation as propagated by [2,3].)

Definition 2.2 The signature of *Minimal Algebra* MA is as follows:

(i) There is a constant $\mathbf{0}$; this denotes inaction, a deadlock state; other names are δ or stop.

(ii) There is a constant $\mathbf{1}$; this denotes termination, a final state; other names are ε, skip or the empty process.

(iii) For each element of the alphabet $\mathcal{A}$ there is a unary operator $a\_$ called *action prefix*; a term $a.x$ will execute the elementary action a and then proceed as x.

(iv) There is a binary operator $+$ called *alternative composition*; a term $x + y$ will either execute x or execute y, a choice will be made between the alternatives.

The constants $\mathbf{0}$ and $\mathbf{1}$ are needed to denote transition systems with a single state and no transitions. The constant $\mathbf{0}$ denotes a single state that is not a final state, while $\mathbf{1}$ denotes a single state that is also a final state.

Definition 2.3 Let $\mathcal{V}$ be a set of *variables*. A *recursive specification* over $\mathcal{V}$ with initial variable $S \in \mathcal{V}$ is a set of equations of the form $X = t_X$, exactly one for each $X \in \mathcal{V}$, where each right-hand side t_X is a term over some signature, possibly containing elements of $\mathcal{V}$. A recursive specification is called finite, if $\mathcal{V}$ is finite.

We find that a finite recursive specification over MA can be seen as a right-linear grammar. Now each finite transition system corresponds directly to a finite recursive specification over MA, using a variable for every state. To go from a term over MA to a transition system, we use *structural operational semantics* [1], with rules given in Table 1.

$$\mathbf{1}\downarrow \qquad a.x \xrightarrow{a} x$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x\downarrow}{x + y\downarrow} \qquad \frac{y\downarrow}{x + y\downarrow}$$

$$\frac{t_X \xrightarrow{a} x \quad X = t_X}{X \xrightarrow{a} x} \qquad \frac{t_X\downarrow \quad X = t_X}{X\downarrow}$$

Table 1
Operational rules for MA and recursion ($a \in \mathcal{A}, X \in \mathcal{V}$).

3 Basic Parallel Processes

The class of basic parallel processes introduced by Christensen in [7] contains processes that can interleave actions of parallel components. In [4], we established that context-free processes can be given by recursive specifications over the Sequential Algebra SA, which extends MA with the *sequential composition* operator $\_ \cdot \_$. In this paper, we give parallel processes, a superset of the basic parallel processes, by recursive specifications over the *Communication Algebra* CA, which extends MA with the *parallel composition* operator $\_ \parallel \_$.

Now, consider the notion of a parallel pushdown automaton. A parallel pushdown automaton is a finite automaton, but at every step it can insert a number of elements into a bag by communicating along port i, or it can remove a single item from the bag by communicating along port o, and take this information into account in determining its next move. Thus, making the interaction explicit, a parallel pushdown automaton is a regular process communicating with a bag. In order to model the interaction between the regular process and the bag, again we use the Communication Algebra CA. We use a particular communication function, that will only synchronize actions $!_c d$ and $?_c d$ (for the same channel $c \in \{i, o\}$ and data element $d \in D$). The result of such a synchronization is denoted $⁉_c d$. Furthermore, CA contains the *encapsulation operator* $\partial_{io}(\_)$, which blocks actions $!_i d$, $?_i d$, $!_o d$ and $?_o d$, and the *abstraction operator* $\tau_{io}(\_)$ which turns all $⁉_i d$ and $⁉_o d$ actions into the

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad \frac{x\downarrow \quad y\downarrow}{x \parallel y\downarrow}$$

$$\frac{x \xrightarrow{?_c d} x' \quad y \xrightarrow{!_c d} y'}{x \parallel y \xrightarrow{\Updownarrow_c d} x' \parallel y'} \qquad \frac{x \xrightarrow{!_c d} x' \quad y \xrightarrow{?_c d} y'}{x \parallel y \xrightarrow{\Updownarrow_c d} x' \parallel y'}$$

$$\frac{x \xrightarrow{a} x' \quad a \notin \{!_c d, ?_c d\}}{\partial_{io}(x) \xrightarrow{a} \partial_{io}(x')} \qquad \frac{x\downarrow}{\partial_{io}(x)\downarrow}$$

$$\frac{x \xrightarrow{\Updownarrow_c d} x'}{\tau_{io}(x) \xrightarrow{\tau} \tau_{io}(x')} \qquad \frac{x \xrightarrow{a} x' \quad a \neq \Updownarrow_c d}{\tau_{io}(x) \xrightarrow{a} \tau_{io}(x')} \qquad \frac{x\downarrow}{\tau_{io}(x)\downarrow}$$

Table 2
Operational rules for CA ($a \in \mathcal{A}$, $c \in \{i, o\}$).

internal action τ. For CA, we extend the operational rules of MA (see Table 1) with operational rules in Table 2.

Consider the following specification:

$$P = \mathbf{1} + P \parallel a.\mathbf{1}.$$

Our first observation is that, by means of the operational rules, we derive an infinite transition system, which moreover is infinitely branching. All the states of this transition system are different in bisimulation semantics, and so this is in fact an infinitely branching process. Our second observation is that this recursive specification has infinitely many different (non-bisimilar) solutions in the transition system model. This is because the equation is *unguarded*, the right-hand side contains a variable that is not in the scope of an action-prefix operator, and also cannot be brought into such a form. So, if there are multiple solutions to a recursive specification, we have multiple processes that correspond to this specification. This causes additional difficulties.

These two observations are the reason to restrict to guarded recursive specifications only. It is well-known that a guarded recursive specification has a unique solution in the transition system model (see [6,5]), and we show later on that this solution is also finitely branching. This restriction leads to our following definition of the basic parallel processes, a subclass of the parallel processes given by recursive specifications over CA.

Definition 3.1 A *basic parallel process* is the bisimulation equivalence class of the transition system generated by a finite guarded recursive specification over the Communication Algebra CA such that the process only interleaves actions and synchronizes termination, but does not allow for communication to happen. That is, only the operational rules on the top line in Table 2 are used.

In this paper, we use equational reasoning to manipulate recursive specifications. Our finite axiomatization of transition systems of CA modulo *rooted branching bisimulation* [9] uses the auxiliary operators $\_ \between \_$ and $\_ \mid \_$ [6,10]. See Table 3. See [3] for an explanation of the axioms.

$x \parallel y = x \between y + y \between x + x \mid y$		$a.(\tau.(x+y)+x) = a.(x+y)$	
$\mathbf{0} \between x = \mathbf{0}$		$x \mid y = y \mid x$	
$\mathbf{1} \between x = \mathbf{0}$		$x \parallel \mathbf{1} = x$	
$a.x \between y = a.(x \parallel y)$		$\mathbf{1} \mid x + \mathbf{1} = \mathbf{1}$	
$(x+y) \between z = x \between z + y \between z$		$(x \parallel y) \parallel z = x \parallel (y \parallel z)$	
$\mathbf{0} \mid x = \mathbf{0}$		$(x \mid y) \mid z = x \mid (y \mid z)$	
$(x+y) \mid z = x \mid z + y \mid z$		$(x \between y) \between z = x \between (y \parallel z)$	
$\mathbf{1} \mid \mathbf{1} = \mathbf{1}$		$(x \mid y) \between z = x \mid (y \between z)$	
$a.x \mid \mathbf{1} = \mathbf{0}$		$x \between \tau.y = x \between y$	
$!_c d.x \mid ?_c d.y = \mathbb{?}_c d.(x \parallel y)$		$x \mid \tau.y = \mathbf{0}$	
$a.x \mid b.y = \mathbf{0}$	*if* $\{a,b\} \neq \{!_c d, ?_c d\}$		
$\partial_{io}(\mathbf{0}) = \mathbf{0}$		$\tau_{io}(\mathbf{0}) = \mathbf{0}$	
$\partial_{io}(\mathbf{1}) = \mathbf{1}$		$\tau_{io}(\mathbf{1}) = \mathbf{1}$	
$\partial_{io}(!_c d.x) = \partial_{io}(?_c d.x) = \mathbf{0}$		$\tau_{io}(\mathbb{?}_c d.x) = \tau.\tau_{io}(x)$	
$\partial_{io}(a.x) = a.\partial_{io}(x)$	*if* $a \notin \{!_c d, ?_c d\}$	$\tau_{io}(a.x) = a.\tau_{io}(x)$	*if* $a \neq \mathbb{?}_c d$
$\partial_{io}(x+y) = \partial_{io}(x) + \partial_{io}(y)$		$\tau_{io}(x+y) = \tau_{io}(x) + \tau_{io}(y)$	

Table 3
Equational theory of CA ($a \in \mathcal{A} \cup \{\tau\}$, $c \in \{i,o\}$).

Besides these axioms we use the Cluster Fair Abstraction Rule for (rooted) branching bisimulation CFAR^b, introduced in [5,12]. For a guarded recursive specification E and the set of abstractions $I \subseteq \mathcal{A}$ we want to abstract from, a subset C of the variables $\mathcal{V}$ is called *a cluster of I in E* if for all $X \in C$, the equation of X in E is of the form

$$X = \sum_{1 \leq k \leq m} i_k.X_k + \sum_{1 \leq j \leq n} Y_j,$$

where $i_1, \ldots, i_m \in I \cup \{\tau\}$, $X_1, \ldots, X_m \in C$, and $Y_1, \ldots, Y_n \in \mathcal{V} - C$. We call the set of variables $\{Y_1, \ldots, Y_n\}$ the *exits* of X, denoted with $U(X)$, and use $U(C)$ to refer to the exit set of the cluster C. The cluster C is called *conservative* if every exit from $U(C)$ is reachable from every variable in the cluster by doing a number of steps from $I \cup \{\tau\}$. Now, CFAR^b is the following rule:

$$\frac{E \text{ guarded} \qquad X \in C \qquad I \subseteq \mathcal{A} \qquad C \text{ is a finite conservative cluster of } I \text{ in } E}{\tau.\tau_I(X) = \tau. \sum_{Y \in U(C)} \tau_I(Y)}.$$

Furthermore, we often use the aforementioned principle that guarded recursive specifications have unique solutions [5].

The given equational theory is sound and ground-complete for the model of transition systems modulo rooted branching bisimulation [9,3]. This is the preferred model we use, but all our reasoning in the following takes place in the equational

theory, so is model-independent provided the models preserve validity of the axioms, unique solutions for guarded recursive specifications and CFAR^b.

Using the axioms, any guarded recursive specification can be brought into *Greibach normal form* [7]:

$$X = \sum_{i \in I_X} a_i.\xi_i \ (+ \mathbf{1}).$$

In this form, every right-hand side of every equation consists of a number of summands, indexed by a finite set I_X (the empty sum is $\mathbf{0}$), each of which is $\mathbf{1}$, or of the form $a_i.\xi_i$, where ξ_i is the parallel composition of a number of variables (the empty multiset is $\mathbf{1}$). For a recursive specification in Greibach normal form, every state of the transition system is given by a multiset of variables just like in [11]. Note that we can take the index sets associated with the variables to be disjoint. As an example, we consider the important basic parallel process *bag*. Suppose D is a finite data set, then we define the following actions in $\mathcal{A}$, for each $d \in D$:

- $?_i d$: insert (push) d into the bag over the input channel i;
- $!_o d$: remove d from the bag over the output channel o.

Now the recursive specification in Greibach normal form is as follows:

$$B = \mathbf{1} + \sum_{d \in D} ?_i d.(B \parallel !_o d.\mathbf{1}).$$

In order to see that the above process indeed defines a bag, define processes B_μ, denoting the bag with contents $\mu \in D^*$, as follows: the first equation for the empty bag, the second for any nonempty bag, with isolated element d and rest bag μ (denoted with $\{d\} \uplus \mu$, but abbreviated with the notation $d\mu$ from here on):

$$\begin{aligned} B_\emptyset &= B, \\ B_{d\mu} &= !_o d.\mathbf{1} \parallel B_\mu. \end{aligned}$$

Then it is straightforward to derive the following equations:

$$\begin{aligned} B_\emptyset &= \mathbf{1} + \sum_{d \in D} ?_i d.B_d, \\ B_{d\mu} &= !_o d.B_\mu + \sum_{e \in D} ?_i e.B_{ed\mu}. \end{aligned}$$

Finally, we define the *forgetful bag*, which can terminate even if it is not empty, as follows:

$$B = \mathbf{1} + \sum_{d \in D} ?_i d.\big(B \parallel (!_o d.\mathbf{1} + \mathbf{1})\big).$$

Note that while the bag is given by a recursive specification of CA, it is a basic parallel process, since no communication is possible between $!_o d$ and $?_i d$ for any $d \in D$.

4 Parallel Pushdown Automata

The main goal of this paper is to prove that every basic parallel process is equal to a regular process communicating with a bag. Thus, if P is any basic parallel process, then we want to find a regular process Q such that

$$P = \tau_{io}(\partial_{io}(Q \parallel B)),$$

where B is a (partially) forgetful bag process specified below.

Without loss of generality, we assume in this section that P is given in Greibach normal form. The data set D we use for our solution is the set of variables $\mathcal{V}$ of P. We call a variable *transparent* if its equation has a $\mathbf{1}$-summand. We denote the set of transparent variables of P with $\mathcal{V}^{+\mathbf{1}}$. Furthermore, we define the conditional process $\mathbf{1}_\xi$ as $\mathbf{1}$ if all variables in set or multiset ξ are transparent and as $\mathbf{0}$ otherwise.

Now, we prove the main theorem by first stating the specification of our solution, then proving necessary lemmas related to this specification before finally giving the main proof.

Theorem 4.1 *For every basic parallel process P there exists a process Q given by a finite guarded recursive specification over MA such that $P = \tau_{io}(\partial_{io}(Q \parallel B_\emptyset)) = [Q \parallel B_\emptyset]_{io}$* [2] *where B is the (partially) forgetful bag.*

Proof Let E be a finite recursive specification of P in Greibach normal form. Now, let F be a recursive specification that defines a parallel pushdown automaton. This specification contains the following equations for every variable $X \in \mathcal{V}$ of the specification E:

$$\hat{X} = \sum_{i \in I_X} a_i.\mathrm{Push}(\xi_i) + \mathbf{1}_X,$$

where $\mathrm{Push}(\xi)$ is recursively defined as

$$\begin{aligned} \mathrm{Push}(\emptyset) &= \mathrm{Ctrl}, \\ \mathrm{Push}(X\xi') &= !_i X.\mathrm{Push}(\xi'). \end{aligned}$$

where X is a variable that is in the original multiset ξ and ξ' is the multiset that is left over when X has been removed.

Additionally, let F contain the following equations of a *partially forgetful bag* and a (regular) finite control:

$$\begin{aligned} B &= \mathbf{1} + \sum_{\substack{V \in \mathcal{V} \\ V \notin \mathcal{V}^{+\mathbf{1}}}} ?_i V.(!_o V.\mathbf{1} \parallel B) + \sum_{\substack{V \in \mathcal{V} \\ V \in \mathcal{V}^{+\mathbf{1}}}} ?_i V.\big((!_o V.\mathbf{1} + \mathbf{1}) \parallel B\big), \\ \mathrm{Ctrl} &= \sum_{V \in \mathcal{V}} ?_o V.(\hat{V} + !_i V.\mathrm{Ctrl}). \end{aligned}$$

[2] From here on, $[p]_{io}$ is used as a shorthand notation for $\tau_{io}(\partial_{io}(p))$.

The specification of each $\hat{X}$ in F mimics the behavior of each X in E by performing the same actions a_i and subsequently inserting each variable of the parallel composition ξ_i in the (partially) forgetful bag B. If X has a $\mathbf{1}$-summand, $\hat{X}$ mimics this by allowing for termination given that the bag is empty or contains transparent variables. Once the insertions are done, the process Ctrl arbitrarily removes a variable V from the bag (which is the multiset of variables that can be executed in parallel at this moment) and executes $\hat{V}$. Note that Ctrl itself does not make a choice. The choice is defered to the point when the first action is performed by one of the variables that Ctrl can remove from the bag.

We interpret the multisets in Greibach normal forms as parallel compositions. In Greibach normal form, every state in P is labeled with a parallel composition of variables ξ. Substituting the Greibach normal form for the variables $X_1, \ldots, X_n$ gives us the following derivation:

$$\begin{aligned}
\xi &= X_1 \parallel \ldots \parallel X_n \\
&= X_1 \between (X_2 \parallel \ldots \parallel X_n) + \ldots \\
&\qquad + X_n \between (X_1 \parallel \ldots \parallel X_{n-1}) + (X_1 \mid X_2 \mid \ldots \mid X_n) \\
&= \sum_{i \in I_{X_1}} a_i.(\xi_i \parallel X_2 \parallel \ldots \parallel X_n) + \ldots \\
&\qquad + \sum_{i \in I_{X_n}} a_i.(\xi_i \parallel X_1 \parallel \ldots \parallel X_{n-1}) + \mathbf{1}_{\{X_1,\ldots,X_n\}} \\
&= \sum_{V \in \xi} \sum_{i \in I_V} a_i.(\xi_i \xi - \{V\}) + \mathbf{1}_\xi.
\end{aligned}$$

Introducing a fresh variable $\overline{P}(\xi)$ for each possible multiset ξ, we obtain the following equivalent infinite recursive specification.

$$\overline{P}(\xi) = \sum_{V \in \xi} \sum_{i \in I_V} a_i.\overline{P}(\xi_i \xi - \{V\}) + \mathbf{1}_\xi.$$

Now that we have an indication of the relationship between the process P and suitable contents of the bag B, we propose the following equation:

$$\overline{P}(\xi) = \sum_{V \in \xi} \left[\hat{V} \between B_{\xi - \{V\}}\right]_{io} + \mathbf{1} \mid B_\xi. \tag{1}$$

Equation 1 expresses the relationship between a state in a basic parallel process, given by a parallel composition of variables, and our regular process communicating with a bag. Given that X is the initial variable of E, we can instantiate the general

case and use the definition of B and the axioms of Table 3 to show that

$$\begin{aligned}\overline{P}(X) &= \left[\hat{X} \between B_\emptyset\right]_{io} + \mathbf{1} \mid B_X \\ &= \left[\hat{X} \between B + B \between \hat{X}\right]_{io} + \mathbf{1} \mid \left((!_o X.\mathbf{1} + \mathbf{1}_X) \mid B_\emptyset\right) \\ &= \left[\hat{X} \between B + B \between \hat{X}\right]_{io} + \mathbf{1}_X \mid B_\emptyset \\ &= \left[\hat{X} \between B + B \between \hat{X}\right]_{io} + \hat{X} \mid B_\emptyset \\ &= \left[\hat{X} \parallel B\right]_{io}.\end{aligned}$$

So, we define $Q = \hat{X}$ and we show that $P = \left[\hat{X} \parallel B_\emptyset\right]_{io}$. This means that we have to prove equation 1 for any multiset of variables ξ. But first we prove a lemma and corollary relating the definition of the conditional process to communication with the partially forgetful bag.

Lemma 4.2 *For all sets or multisets of variables ξ it holds that $\mathbf{1} \mid B_\xi = \mathbf{1}_\xi$.*

Proof By induction over the contents of ξ.

(i) If $\xi = \emptyset$, then $\mathbf{1} \mid B_\emptyset = \mathbf{1} \mid B = \mathbf{1} = \mathbf{1}_\emptyset$.

(ii) If $\xi = X\xi'$, then
   (a) if $X \in \mathcal{V}^{+\mathbf{1}}$, then $\mathbf{1} \mid B_{X\xi'} = \mathbf{1} \mid \left((!_o X.\mathbf{1} + \mathbf{1}) \parallel B_{\xi'}\right) = \mathbf{1} \mid !_o X.\mathbf{1} + \mathbf{1} \mid B_{\xi'} = \mathbf{1} \mid B_{\xi'}$. Because, by induction hypothesis, $\mathbf{1} \mid B_{\xi'} = \mathbf{1}_{\xi'}$, we have that $\mathbf{1} \mid B_\xi = \mathbf{1}_{\xi'}$ given that $X \in \mathcal{V}^{+\mathbf{1}}$ and therefore $\mathbf{1} \mid B_\xi = \mathbf{1}_\xi$.
   (b) if $X \notin \mathcal{V}^{+\mathbf{1}}$, then $\mathbf{1} \mid B_{X\xi'} = \mathbf{1} \mid (!_o X.\mathbf{1} \parallel B_{\xi'}) = \mathbf{1} \mid !_o X.\mathbf{1} \mid B_{\xi'} = \mathbf{0} = \mathbf{1}_\xi$ because ξ contains the non-transparent variable X.

□

Corollary 4.3 *For all sets or multisets of variables ξ and every variable X it holds that $\mathbf{1}_X \mid B_{\xi - \{X\}} = \mathbf{1}_\xi$.*

Proof By Lemma 4.2, we have that $\mathbf{1}_X \mid B_{\xi-\{X\}} = \mathbf{1} \mid B_X \mid B_{\xi-\{X\}}$. By the definition of B, it follows that $B_X \mid B_{\xi-\{X\}} = B_\xi$. Therefore, again by Lemma 4.2, $\mathbf{1}_X \mid B_{\xi-\{X\}} = \mathbf{1} \mid B_\xi = \mathbf{1}_\xi$. □

Now we prove the following lemma, which is crucial for the main proof. This lemma expresses that if the finite control is at a point where it can choose a variable from the bag, it does not make the actual choice. The choice is determined by the first action that is performed by a candidate variable. It also shows that when this has happened, this particular variable has also been removed from the bag.

Lemma 4.4 *For any non-empty multiset ξ contained in a bag, it holds that $\tau.[\mathrm{Ctrl} \parallel B_\xi]_{io} = \tau.\sum_{V \in \mathcal{V}} \left[\hat{V} \parallel B_{\xi-\{V\}}\right]_{io}$.*

Proof We use the following definitions: $C = \partial_{io}(\mathrm{Ctrl} \parallel B_\xi)$, $Y_V = \partial_{io}(\hat{V} \parallel B_{\xi-\{V\}})$,

and $X_V = ?_i V.C + Y_V$ for all $V \in \mathcal{V}$. Let us now consider C:

$$\begin{aligned}
C &= \partial_{io}(\sum_{V\in\mathcal{V}} ?_o V.(!_i V.\mathrm{Ctrl} + \hat{V}) \parallel B) \\
&= \sum_{V\in\mathcal{V}} ?_o V.\partial_{io}((!_i V.\mathrm{Ctrl} + \hat{V}) \parallel B_{\xi-\{V\}}) \\
&= \sum_{V\in\mathcal{V}} ?_o V.\partial_{io}(!_i V.\mathrm{Ctrl} \between B_{\xi-\{V\}} + \hat{V} \between B_{\xi-\{V\}} + \hat{V} \mid B_{\xi-\{V\}}) \\
&= \sum_{V\in\mathcal{V}} ?_o V.\big(?_i V.\partial_{io}(\mathrm{Ctrl} \parallel B_\xi) + \partial_{io}(\hat{V} \parallel B_{\xi-\{V\}})\big) \\
&= \sum_{V\in\mathcal{V}} ?_o V.(?_i V.C + Y_V) \\
&= \sum_{V\in\mathcal{V}} ?_o V.X_V.
\end{aligned}$$

If we apply the CFAR^b rule on the specification containing $C = \sum_{V\in\mathcal{V}} ?_o V.X_V + \mathbf{0}$ and $X_V = ?_i V.C + Y_V$ for each $V \in \mathcal{V}$, which forms a cluster of $\{?_i d, ?_o d\}$ in this specification, we obtain: $\tau.\tau_{io}(C) = \tau.\sum_{V\in\mathcal{V}} \tau_{io}(Y_V)$. Hence,

$$\tau.\,[\mathrm{Ctrl} \parallel B_\xi]_{io} = \tau.\tau_{io}(C) = \tau.\sum_{V\in\mathcal{V}} \tau_{io}(Y_V) = \tau.\sum_{V\in\mathcal{V}} \left[\hat{V} \parallel B_{\xi-\{V\}}\right]_{io}.$$

□

Now that all prerequisites are in place, we can deal with the main proof which requires us to prove the following statement:

$$\overline{P}(\xi) \stackrel{?}{=} \sum_{V\in\xi} \left[\hat{V} \between B_{\xi-\{V\}}\right]_{io} + \mathbf{1} \mid B_\xi$$

First, apply the definition of $\hat{V}$ and get rid of the left merges.

$$= \sum_{V\in\xi} \sum_{i\in I_V} a_i.\left[\mathrm{Push}(\xi_i) \parallel B_{\xi-\{V\}}\right]_{io} + \mathbf{1} \mid B_\xi$$

Perform $|\xi_i|$ pushes by repeatedly applying the definition of $\mathrm{Push}(\xi)$.

$$= \sum_{V\in\xi} \sum_{i\in I_V} a_i.\tau^{|\xi_i|}.\left[\mathrm{Ctrl} \parallel B_{\xi_i\xi-\{V\}}\right]_{io} + \mathbf{1} \mid B_\xi$$

Remove all but one τ-step that follows a_i or introduce one τ-step if ξ_i is empty and apply Lemma 4.4 on $\tau.\left[\mathrm{Ctrl} \parallel B_{\xi_i\xi-\{V\}}\right]_{io}$.

$$= \sum_{V\in\xi} \sum_{i\in I_V} a_i.\tau.\sum_{W\in\xi_i\xi-\{V\}} \left[\hat{W} \parallel B_{\xi_i\xi-\{V,W\}}\right]_{io} + \mathbf{1} \mid B_\xi$$

Remove the τ-step and perform expansion on the merge operator and remove the summand $\left[B_{\xi_i\xi-\{V\}} \between \hat{W}\right]_{io}$ since its left-hand side cannot perform any non-encapsulated action.

$$= \sum_{V\in\xi}\sum_{i\in I_V} a_i.\Big(\sum_{W\in\xi_i\xi-\{V\}} \Big(\left[\hat{W} \between B_{\xi_i\xi-\{V,W\}}\right]_{io} + \hat{W} \mid B_{\xi_i\xi-\{V,W\}}\Big)\Big) + \mathbf{1} \mid B_\xi$$

Now, consider the summand $\hat{W} \mid B_{\xi_i\xi-\{V,W\}}$. Since $\hat{W}$ cannot perform any action, only the summand $\mathbf{1}_W$ remains of the specification of $\hat{W}$.

$$= \sum_{V\in\xi}\sum_{i\in I_V} a_i.\Big(\sum_{W\in\xi_i\xi-\{V\}} \Big(\left[\hat{W} \between B_{\xi_i\xi-\{V,W\}}\right]_{io} + \mathbf{1}_W \mid B_{\xi_i\xi-\{V,W\}}\Big)\Big) + \mathbf{1} \mid B_\xi$$

Because $\mathbf{1}_W \mid B_{\xi_i\xi-\{V,W\}} = \mathbf{1}_{\xi_i\xi-\{V\}}$ by Corollary 4.3 and $\mathbf{1}_{\xi_i\xi-\{V\}}$ does not depend on W, we can move it outside of the summation.

$$= \sum_{V\in\xi}\sum_{i\in I_V} a_i.\Big(\sum_{W\in\xi_i\xi-\{V\}} \left[\hat{W} \between B_{\xi_i\xi-\{V,W\}}\right]_{io} + \mathbf{1}_{\xi_i\xi-\{V\}}\Big) + \mathbf{1} \mid B_\xi$$

Use the definition of $\overline{P}(\xi_i\xi - \{V\})$ and apply Lemma 4.2 on $\mathbf{1} \mid B_\xi$.

$$= \sum_{V\in\xi}\sum_{i\in I_V} a_i.\overline{P}(\xi_i\xi - \{V\}) + \mathbf{1}_\xi.$$

This concludes our proof that there exists a recursive specification over CA that, in parallel with a partially forgetful bag, is equivalent to a basic parallel process P. □

A corollary of this theorem strengthens the result found in [8], that basic parallel processes have finite branching. In fact, the branching is bounded (i.e. there is a fixed maximum branching for all reachable states).

Corollary 4.5 *Every basic parallel process has bounded branching.*

Proof By Theorem 4.1 there exists a regular process Q (given by a finite guarded recursive specification over MA) for every basic parallel process P such that $P = [Q \between B_\emptyset]_{io}$. Because Q is regular, it is boundedly branching. The process B is also boundedly branching. Because the parallel composition leads to the Cartesian product of both boundedly branching components plus the result of communication [5], P is also boundedly branching. □

5 Concluding Remarks

We have proved that every basic parallel process is rooted branching bisimilar to a regular process communicating with a bag. A regular process communicating with a bag can be seen as a parallel pushdown automaton, and so this result extends the result of [11] by adding the process $\mathbf{1}$ and making the internal communication

explicit. As a result, we can see the bag as the prototypical basic parallel process. As a corollary, we established that every basic parallel process has bounded branching.

This is in contrast to the situation with context-free processes. We saw in [4] that context-free processes can show unbounded branching. For a context-free process with unbounded branching, we cannot show it is rooted branching bisimilar to a regular process communicating with a stack. We could only show this in contrasimulation. Here, for basic parallel processes, the situation is simpler, and we can establish the full result in rooted branching bisimulation.

The reverse direction, to see if any regular process communicating with a bag is actually a basic parallel process is open as far as we know. Of course, in order to achieve this result, we should allow τ-steps in the definition of basic parallel process, but that is no problem as long as we make sure we retain guardedness. Note that [11] shows the reverse direction is not true in the absence of $\mathbf{1}$, but here, with $\mathbf{1}$, it might still be true.

In addition, it is open whether the result of [8] that bisimulation equivalence is decidable for basic parallel processes is still valid with the addition of $\mathbf{1}$.

An interesting extension of basic parallel processes is allowing communication in the definition of processes. Probably, expressive power will increase, but we do not know examples of processes that can be defined with the addition of communication but not without communication.

Finally, note that the addition of $\mathbf{1}$ allows termination exactly when a bag is empty. This check on emptiness is not possible without $\mathbf{1}$. This is different from the situation with a stack, where a check on empty is also possible in an ACP-style language.

Having looked at stacks and bags, it is interesting to look at queues next. Thus, it is interesting to see which set of processes can be realized as a regular process communicating with a queue.

Acknowledgement

The research of Van Tilburg was supported by the project "Models of Computation: Automata and Processes" (nr. 612.000.630) of the Netherlands Organization for Scientific Research (NWO).

References

[1] Aceto, L., Fokkink, W.J. and Verhoef, C., *Structural operational semantics*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 197–292.

[2] Baeten, J.C.M., Basten, T. and Reniers, M.A., "Process Algebra: Equational Theories of Communicating Processes," Cambridge University Press, 2008.

[3] Baeten, J.C.M. and Bravetti, M., *A ground-complete axiomatization of finite state processes in process algebra*, in: M. Abadi and L. de Alfaro, editors, *Proceedings of CONCUR 2005*, number 3653 in LNCS (2005), pp. 246–262.

[4] Baeten, J.C.M., Cuijpers, P.J.L. and Tilburg, P.J.A. van, *A context-free process as a pushdown automaton*, in: F. v. Breugel and M. Chechik, editors, *Proceedings of CONCUR 2008*, number 5201 in LNCS (2008), pp. 98–113.

[5] Baeten, J.C.M. and Weijland, W.P., "Process Algebra," Cambridge University Press, 1990.

[6] Bergstra, J.A. and Klop, J.W., *Process algebra for synchronous communication*, Information and Control **60** (1984), pp. 109–137.

[7] Christensen, S., "Decidability and decomposition in process algebras," Ph.D. thesis, University of Edinburgh (1993).

[8] Christensen, S., Hirshfeld, Y. and Moller, F., *Bisimulation equivalence is decidable for basic parallel processes*, in: E. Best, editor, *Proceedings of CONCUR 1993*, number 715 in LNCS (1993), pp. 143–157.

[9] Glabbeek, R.J. van and Weijland, W.P., *Branching time and abstraction in bisimulation semantics*, Journal of the ACM **43** (1996), pp. 555–600.

[10] Moller, F., *The importance of the left merge operator in process algebras*, in: M. Paterson, editor, *Proceedings of ICALP'90*, number 443 in LNCS (1990), pp. 752–764.

[11] Moller, F., *Infinite results*, in: U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, number 1119 in LNCS (1996), pp. 195–216.

[12] Vaandrager, F.W., *Verification of two communication protocols by means of process algebra*, Technical Report CS-R8608, CWI, Amsterdam (1986).